FCDS Florida Cancer Data System

Florida Statewide Cancer Registry

A Feasibility Study of the Python Package `splink` for Probabilistic Record Linkage (2023 Monograph Supplement)

Anders Alexandersson

06/30/2023

This 2023 monograph supplement describes a feasibility study at the Florida Cancer Data System (FCDS) of the Python package `splink` (Linacre et al. 2022) for probabilistic record linkage (PRL). Currently, the FCDS uses the R package `fastLink` (Enamorado, Fifield, and Imai 2019; Enamorado 2021) for PRL. The package `splink` is used on large test data with 1 million * 1 million records (observations) from Abraham Flaxman at the University of Washington. The results suggest that it is feasible to replace `fastLink` with `splink` in linkage data requests, because `splink` was 50 times faster and more accurate on the test data. However, for now, `fastLink` remains better overall than `splink` by being more user-friendly. `fastLink` has a template, much simpler syntax and better Markdown support, and linkage results are summarized with tables that are easy to understand.

1 Introduction

The Florida Cancer Data System (FCDS) uses the R package fastLink (Enamorado, Fifield, and Imai 2019; Enamorado 2021) for Fellegi-Sunter probabilistic record linkage (PRL). PRL is usually described as one step in a data cleaning pipeline with these four steps (e.g., Binette and Steorts 2022): 1) attribute alignment, 2) blocking, 3) PRL, and 4) canonicalization.

The FCDS has tested the performance of Match*Pro version 1.6.2 against `fastLink` version 0.6. `Match*Pro` with the Expectation Maximization (EM) algorithm resulted in 1-2% more missed matches and `Match*Pro` without the EM algorithm resulted in 1-2% wrong matches. Therefore, the FCDS prefers `fastLink` over `Match*Pro`.

The FCDS has developed two R Markdown document templates for easier use of `fastLink` within the RStudio integrated development environment (IDE) by Posit. One such template, "the FCDS template", is for FCDS use only. The other template, "the `fastLink` template", is for possible non-FCDS use for three reasons: 1) to help Florida Department of Health use `fastLink`, for example in data request 329, 2) to easier get help from the developers, and 3) to contribute our use expertise to the cancer community, for example in the annual NAACCR conference. The main advantage of the templates is easier reproducible reports in PDF of `fastLink` results. The renv package helps you to create **r**eproducible **env**ironments for your R projects.

PRL can be more complicated than deterministic record linkages because of the requirement to estimate separate match probabilities for matches ("m-probability") and for non-matches ("u-probability"). The main advantage of PRL is the ability to incorporate poor-quality matching variables such as address, date of birth, gender, and name (ADGN). Originally, ADGN was used for rule-based deterministic record linkage (Ansolabehere and Hersh 2017). A link is defined as a declared match. Record linkage is performed with two different objectives: matching of entire databases ("matchability") and identification of individuals ("identifiability") (Ansolabehere and Hersh 2017). Usually, the record linkage process is the same for both objectives. In practice, matchability focuses on increasing the total number of matches whereas identifiability focuses on reducing the wrong matches.

This 2023 monograph describes an FCDS feasibility study of the Python package splink (Linacre et al. 2022) for PRL. The monograph examines three possibly feasible and viable uses for `splink` at the FCDS: 1) replace `fastLink` for linkage data requests, 2) reduce the amount of manual review for `Match*Pro` de-duplication, and 3) replace real-time deterministic record linkages with real-time PRL. The focus is on the technological feasibility of replacing `fastLink` for linkage data requests. Section 2 will discuss a feasibility framework and reproducibility, sections 3-6 will discuss the four processing steps, section 7 will discuss the feasibility for the three uses, and section 8 will conclude.

2 A TELOS feasibility framework for `splink`, and reproducibility

Feasibility is the possibility and ability for something to be done. Viability is that something's ability to survive. The Wikipedia article on feasibility study recommends the TELOS framework, an acronym for five key components. There is no established feasibility framework specifically for PRL. When TELOS is applied to PRL, it is argued here, the equivalent acronym is "ASTUE" as in Table 1.

Table 1. The "ASTUE" Feasibility Framework for PRL

Feasibility Dimension	TELOS Components	"ASTUE" Components	Example Categories
1	**T**echnological	**A**ccuracy	EM, Active Learning
2	**E**conomic	**S**peed	Blocking, Size
3	**L**egal	**T**ransparency	Freedom, Openness
4	**O**perational	**U**ser-friendliness	GUI, Template
5	**S**cheduling	**E**xtensibility	API, Reporting

In general, `Match*Pro` is best for beginners, `fastLink` is best for R users, and `splink` is best for Python users. Pre-processing `Match*Pro` is best with `NAACCR*Prep` or `File*Pro`, and post-processing `Match*Pro` is best with SAS. Pre- and post-processing `splink` or `fastLink` do not require any additional skills.

The technological performance is typically determined by **"accuracy"** measures. Accuracy is a combination of expected correct matches (true positives, TP), missed matches (false negatives, FN), wrong matches (false positives, FP), and missed matches (false negatives, FN). It is common to illustrate these four possible outcomes of a classification in a confusion table (also known as "error matrix"). `fastLink` easily shows the confusion table but `Match*Pro` and `splink` do not. Most linkages are imbalanced by having mostly TN. The author recommends the Matthews correlation coefficient to adjust for the balance ratios of the four confusion table categories (TP, TN, FP, and FN). At the FCDS, expected FN are acceptable but expected FP are not acceptable. Therefore, the FCDS focus is on identfiability, not on matchability.

In general, frequency-based algorithms, such as the `Match*Pro` default, are the least accurate (but useful for matchability and blocking), EM algorithms such as `fastLink` and `splink` are more accurate, and advanced algorithms such as the `fastLink` development version with "Active Learning" (Enamorado 2019) or the Bayesian `dblink` (Enamorado and Steorts 2020) are the most accurate. In the record linkage literature, a frequency-based algorithm is known as EpiLink (Contiero et al. 2005). Link Plus with the Direct Method, the R package RecordLinkage with the function epiWeights and the free (open-source) Java software Mainzelliste are three other record linkage software which also use a frequency-based EpiLink-like algorithm.

The EM algorithm, which is an iterative maximum likelihood approach, should be similarly accurate in `Match*Pro`, `fastLink` and `splink`. `splink` can be more accurate than `Match*Pro` and `fastLink` by enabling more complex models. `splink` produces a wide variety of interactive outputs for quality control, for example waterfall charts. A `splink` blog post illustrates that more complex models often outperform simpler models.

The three software are all "free" as in zero cost. In general, Python users are more costly to pay for than R users who are most costly than people who cannot use Python or R. The economic costs are time. As Benjamin Franklin said, "Remember, time is money". Therefore, we want **speed**. The total time includes pre-processing, blocking, PRL, and post-processing. The package `splink` is faster than `fastLink` by using a SQL backend. The default backend is DuckDB. `splink` is capable of linking a million records on a modern desktop in around a minute. See the topic guide for more info.

The key legal component is **transparency**. Transparency is about software freedom and openness of development. There are four types of software freedom:

- The freedom to run the program as you wish, for any purpose.
- The freedom to study how the program works, and change it as you wish.
- The freedom to redistribute copies so you can help others.
- The freedom to distribute copies of your modified versions to others.

In terms of freedom, `Match*Pro` is non-free (proprietary, closed source) whereas `fastLink` and `splink` are free. Python and `splink` use BSD-style licenses (PSF and MIT respectively). R and `fastLink` uses a GPL-style license (GPL-3). The main difference is that BSD-style licenses allow distributing a closed-source version, modified or not, whereas GPL-style licenses do not allow it. BSD-style licenses are often referred to as "open source" as opposed to "free". BSD-style licenses are free for the developers whereas GPL-style licenses are free for the users. `fastLink` is less transparent than `splink` by having a much larger gap between the latest public and private development version, as evidenced by publications (e.g., Enamorado 2021).

The operational or organizational component is about the **user-friendliness** *from the user's perspective*. For the author, `fastLink` is more user-friendly than `splink`. `fastLink` has a template, much simpler syntax, and linkage results are summarized with tables (especially the confusion table) that are easy to understand.

The scheduling component is about time estimates for **extensibility**. Two aspects of extensibility are Application Programming Interface (API) and reporting capabilities. `Match*Pro` does not have an API for data cleaning or reporting capabilities but `fastLink` and `splink` have both. The next version of `Match*Pro` is expected to have a faster EM algorithm.

We used the RStudio instructions for installing Python on a Windows desktop. Specifically, we disabled App execution aliases, and we used pyenv-win. We used Quarto version 1.3.361 which was released on May 26, 2023. Quarto is a new IDE for Python, R, Julia, and Observable JS. Quarto Markdown (.qmd) files are plain text files. Quarto works with VS Code, JupyterLab, RStudio, and with regular text editors. See the tutorial Get Started. RStudio for Python requires the `reticulate` package. You need to both install it (with the command `install.packages("reticulate")`) and use it (with the command `reticulate::repl_python()`); for details, see the Posit support article.

The main alternative to `pyenv-win` is the standard Python installer from www.python.org/downloads/. Python installation is infamously difficult because of varying project environments, for a funny example see XKCD comic figure 1987. The standard installer, unlike `pyenv-win`, uses the so called Python launcher `py.exe`. "Add a guide on how to setup Python on Windows to work well with Quarto" is Quarto issue 4737 to be resolved in Quarto 1.4.

Reproducibility is a key component in any study, especially for PRL since accuracy is critical. For `fastLink`, the R package `renv` provides a reproducible environment. A reproducible environment for `splink` is using `venv`. The main advantage of using Quarto for this feasibility study is easier reproducible reports in PDF of `splink` results. Other advantages of using Quarto are multi-language support (Python, R, Julia, and Observable JS) and advanced reporting features. See the Markdown file `readme.md` for the actual code used to create a reproducible environment for `splink`. The file describes a two-step process:

1. Create a virtual environment using venv. `venv` is included in the Python standard library and it requires no additional installation.

2. Install the package list with the file requirements.txt. The user documentation could be improved, see discussion 1295 and issue 1031.

The FCDS `splink` project comes with these nine files to be used for a reproducible environment:

1. `readme.md` – Read this file first. It assumes Python and Quarto being installed.
2. `requirements.txt` – A plain text file of the required Python packages, similar to `renv.lock` for R's `renv`.
3. `logo.jpg` – A JPG file of the FCDS logo.
4. `dr_linkage.bib` – The bibliography file for references.
5. `synth_prl_no_noise_2022_0420.csv` – A clean dataset from Abraham Flaxman.
6. `synth_prl_w_noise_2022_04_20.csv` – A noisy dataset from Abraham Flaxman.
7. `splink_monograph_2023_supplement.qmd` – The Quarto Markdown file.
8. `splink_monograph_2023_supplement.pdf` – The PDF output file (for printing).
9. `splink_monograph_2023_supplement.docx` – The MS Word output file (for editing).

You can customize output in Quarto, for example to add page numbers in MS Word output. An HTML output file is not provided because the HTML format actually creates many files, including JS, CSS and WOFF. However, HTML is the best format for interactive use because it enables interactive charts. To render all formats, since the document is not part of a Quarto website, the easiest is to use `quarto render splink_monograph_2023_supplement.qmd` on the command line (in the RStudio "Terminal" tab). Quarto for Python has some issues with Windows shared drives, and especially for PDF output. See RStudio issue 147982. The FCDS uses the SMB (rather than the NFS) file-sharing protocol. For SMB shares, see Quarto issue 2380. To workaround the Quarto issue, we created a new Quarto project on "V:\Testing\Quarto" using `File -> New Project...`. Another earlier workaround was to use Quarto on "C:\temp" on Windows 10. For easier reference, we added line numbers to the Python code and code annotation which are new features in Quarto 1.3.

In addition to Quarto, a requirement here for using `splink` is Vega-Lite "charts". In general, charts are interactive whereas graphs are not. Starting with the current version 3.9.2, `splink` uses the package Altair 5.0.1 for rendering Vega-Lite charts because the rendering logic is complex, especially for converting to PDF. This allows the charts to be saved as, for example, `chart.save("myfile.png")`. For details, see pull 1315 and discussions 926 and 1044.

. `splink` uses Vega-Lite charts for three reasons:

- Their interactive features (popups, pan, zoom, crossfilter etc.)
- Ability to easily integrate into web-ready outputs
- They are supported by multiple languages (Python, R, Julia etc.)

The `splink` developers, led by Robin Linacre, use Jupyter Notebooks (.ipynb) files and the package nbconvert for reproducible PDF outputs. RStudio IDE should ship with the latest available stable version of Quarto at time of IDE release (see Quarto issue 2374).

3 Step 1: Attribute alignment

Attribute alignment is data pre-processing for record linkage, up until but not including blocking. In the two FCDS `fastLink` templates, the R Markdown files (`dr000.rmd` and `fltest.rmd` respectively) use the `source()` function to call separate R files with the source code for the data cleaning steps. In Python, in general, the recommended way instead is to treat the file as a module and use `import` *filename*. A module is a Python file that ends in `.py`, and has a name that is importable. A package is a directory that has a file named `__init__.py` in it. Quarto has an equivalent to the R `source()` function in the extension include-code-files. In this feasibility study, we have all Python code and all text in a single, self-contained Quarto Markdown (.qmd) file.

For linkage data requests at the FCDS, the requestor provides a fixed-width file and the FCDS data are extracted and saved as a fixed-width file, for instance, from an Oracle database. The Python package `pandas` can read fixed-width files with the function read_fwf. The Python driver python-oracledb provides a Python API to access an Oracle database. For this feasibility study of `splink`, please see sample code below for how to read in fictitious (synthetic) comma-separated values (CSV) data from Abraham Flaxman at the University of Washington. The datasets are not yet publicly available. However, several other related datasets for PRL are now available from the package `pseudopeople`.

Python indexing begins with 0. Therefore the last observation is 999,999. For example, one difference is for `unique_id` 999998; `ssn` of the noisy data is missing (`NaN`). To show this in PDF, the code block needs to be rendered twice in RStudio if using the "Render" button. In `splink` 3.9.1, to get PDF output in one pass we used the RStudio terminal command `quarto render test_splink.qmd --to pdf`. In the current version `splink` 3.9.2, the issue seems resolved. Knitr and Jupyter are Quarto "binding engines" to render the R and Python computations, respectively. Knitr assumes you want all output whereas Jupyter assumes you only want the last output. For Knitr, results: hold is possible which would hold the results into a chunk for more compact output. For Jupyter, we display all results by changing the `InteractiveShell` option to “all”; see notation 2. The code below opens both datasets and outputs the last 5 records of each dataset:

```
# Step 0: Input data
import time                                                          ①
import pandas as pd
import warnings
from altair_saver import save
from IPython.core.interactiveshell import InteractiveShell

warnings.filterwarnings('ignore')                                    ②
InteractiveShell.ast_node_interactivity = "all"

pd.options.display.max_columns = 0                                   ③
pd.set_option('display.expand_frame_repr', False)
pd.set_option('max_colwidth', 15)

df_noise = pd.read_csv("synth_prl_w_noise_2022_04_20.csv")           ④
df_noise["unique_id"] = df_noise["unique_id"].astype(str)
# warnings.filterwarnings('ignore')
df_orig = pd.read_csv("synth_prl_no_noise_2022_04_20.csv")
df_orig["unique_id"] = df_orig["unique_id"].astype(str)
```

① import modules
② display all output, which is not the Jupyter default
③ do not cutoff the text at right-margin for PDF
④ read the data

Table 2. List of the Last 5 records of the two Linkage Datasets

```
# Step 1: Attribute alignment
df_noise.tail()                                                      ①
df_orig.tail()                                                       ②
```

① Noisy dataset – last 5 rows
② Original dataset – last 5 rows

	unique_id	first_name	last_name	sex	dob	address	zip	ssn
999995	999995	Osvaldo	Foster	1	1974-08-15	358 Lam Hig...	33864	404-53-8579
999996	999996	Zaria	Alwine	2	2011-09-08	9866 Hammon...	34105	029-58-6474
999997	999997	Michael	Chen	1	2007-06-18	243 Andhny ...	33408	074-43-4290
999998	999998	Ronald	Carter	1	1961-10-12	83498 Myers...	32941	NaN
999999	999999	Jennifer	Delgado	2	1973-05-04	10291 Peter...	33410	299-23-6181

	unique_id	first_name	last_name	sex	dob	address	zip	ssn
999995	999995	Osvaldo	Foster	1	1974-08-15	358 Lam Hig...	33864	404-53-8579
999996	999996	Zaria	Alwine	2	2011-09-08	9866 Hammon...	34105	029-58-6474
999997	999997	Michael	Chen	1	2007-06-18	243 Anthony...	33408	074-43-4290
999998	999998	Ronald	Carter	1	1961-10-12	83498 Myers...	32941	381-84-4367
999999	999999	Jennifer	Delgado	2	1973-05-04	10291 Peter...	33410	299-23-6181

The FCDS received the test data on 4/21/2022. The original dataset has no errors, and the noisy dataset has these errors:

- ssn with blanks (15%), e.g., for `unique_id` 999998
- address with changes (10%)
- addresses with typographic errors (5%), e.g., for `unique_id` 999997
- first name typographic errors (5%)
- last name typographic errors (5%)

The `splink` data prerequisites are aligned attributes. For the FCDS using `fastLink`, the most computing time for attribute alignment is to create double metaphone versions of first name (variable `first_name`) and last name (variable `last_name`). For `splink`, double metaphone is available from the Python package `metaphone`. For a related `splink` discussion, see issue 461.

The typical reason for using double metaphone versions of first name and last name in `fastLink` is to free up a string distance method for other string linkage variables. There are three string distance methods in `fastLink`: "jw" Jaro-Winkler (default), "jaro" Jaro, and "lv" Levenshtein edit. Typically, this author uses the Levenshtein edit distance measure to calculate partial matches of Social Security Number (SSN). The Levenshtein edit distance measure makes no sense for first name and last name, whereas the Jaro-Winkler and Jaro distance measures make no sense for SSN. Only at most one string distance method may be chosen in the current version 0.6 of `fastLink`. This is a significant limitation of `fastLink`. In contrast, `splink` has no such limitation. That is, `splink` may use any string distance method for any string variable which could potentially improve matches. `splink` (since version 3.9.1) and `Match*Pro` have the Damerau-Levenshtein comparator which can be used for more precise partial matching of, for example, SSN.

Three other pre-processing packages than `metaphone` are: probablepeople for parsing names, usaddress for parsing U.S. addresses and the Python Record Linkage Toolkit for data preprocessing for cleaning and phonetic encoding.

In the `fastLink` template for step 1, attribute alignment, there are two additional tables:

- Table 1: The dimensions of the datasets
- Table 2: The missingness of the cleaned datasets, by variable

Both tables are created using R inline code in Knitr. There is currently no support for Python inline code in Jupyter. The recommended workaround is to use Display(Markdown()) which is inelegant. Therefore, Tables 1 and 2 in the `fastLink` template are not created here.

An FCDS issue for both `fastLink` and `splink` is that there are no readily available standardized names and addresses in the FCDS database. Overall, the most time consuming part for implementing step 1, attribute alignment, in Python for FCDS linkage data requests likely will be to convert the "generic user missing" value 9 (for example, `999999999` for Social Security Number or `99999` for Zip Code) to missing in Pandas because no such code exists. There are no plans for `splink`, mostly due to resource constraints, to create code for attribute alignment.

4 Step 2: Blocking

Because the main claimed advantage with `splink` is speed, it is important to time the code. `Match*Pro`, `fastLink` and `splink` all allow "AND (conjunctive)" blocking. Only `fastLink` does not allow "OR (disjunctive)" blocking. Only `splink` distinguishes between blocking rules for prediction vs estimation (training). In `splink`, blocking rules are specified as SQL expressions. The `fastLink` developers are working on "probabilistic blocking" (Enamorado and Steorts 2020) where PRL is used also for blocking.

It is difficult to analyze the blocking results in `Match*Pro`. It does not save the blocking results separately. The `Match*Pro` log file does not specify which variables were used in the blocking step. In contrast, the `fastLink` result of blocking is an R object which you can examine. For `splink`, you can similarly use a linker object to compute the number of comparisons that will be generated by a blocking rule. A `splink` tutorial on blocking clarifies what blocking rules are, and how to choose good rules.

The three last `splink` settings in the code below are, respectively, to enable inspecting matches, retain intermediate calculation columns, and the convergence tolerance for the EM algorithm. The EM convergence default is 0.0001. Setting a higher default such as 0.1 reduces the computation time, and it can help the EM algorithm to converge (avoid "numerical underflow") because we expect at least 10% matches. The `Match*Pro` default is 0.05. In general, the `splink` default 0.0001 is good if we have no expectation for matches as in the first run. This linkage has no issues with numerical underflow.

```
# Step 2: Blocking
start_time = time.time()                                                  ①

from splink.duckdb.linker import DuckDBLinker                             ②
from splink.duckdb import comparison_library as cl

settings = {                                                              ③
    "link_type": "link_only",                                             ④
    "blocking_rules_to_generate_predictions": [                           ⑤
        "l.first_name = r.first_name and l.last_name = r.last_name",
        "l.zip = r.zip and substr(l.last_name, 1,2) = substr(r.last_name, 1,2)",
        "l.zip = r.zip and substr(l.first_name, 1,2) = substr(r.first_name, 1,2)",
        "l.last_name = r.last_name and l.dob = r.dob",
        "l.first_name = r.first_name and l.dob = r.dob",
    ],
    "comparisons": [                                                      ⑥
        cl.jaro_winkler_at_thresholds("first_name", [0.9, 0.7],
            term_frequency_adjustments=True),
        cl.jaro_winkler_at_thresholds("last_name", [0.9, 0.7],
            term_frequency_adjustments=True),
        cl.levenshtein_at_thresholds("dob", [1,2],
            term_frequency_adjustments=True),
        cl.exact_match("sex"),
        cl.levenshtein_at_thresholds("zip", [1,2],
            term_frequency_adjustments=True),
        cl.exact_match("ssn"),
    ],
    "retain_matching_columns": True,                                      ⑦
    "retain_intermediate_calculation_columns": True,
    "em_convergence": 0.01
}
```

① start to time the code
② DuckDB-specific code, see Choosing a backend
③ Define a `splink` model with a settings dictionary
④ Link type: Linkage *without* de-duplication.
⑤ Blocking for prediction: Ensure comparisons are generated for all true matches
⑥ Customise comparisons using the ComparisonLibrary and Term-Frequency adjustments
⑦ See guide to `splink` settings for more details

5 Step 3: Record linkage

Most of Splink's functionality can be accessed by calling methods (functions) on the linker object, such as `linker.predict()`, `linker.profile_columns()` etc.

```
# Step 3: Record Linkage
linker = DuckDBLinker([df_orig, df_noise], settings,                    ①
     input_table_aliases=["no_noise", "with_noise"])
```

① The linker object

For example, the `splink` function `linker.missingness_chart().spec` creates a missingness chart. In contrast to tables 1 and 2 in the `fastLink` template, the missingness chart/graph for `splink` is either for the linked dataset (default) or a specified input dataset. It tells you how much missingness each linkage variable has. The `splink` missingness chart is difficult to work with, for example to change the title. In general, over approximately 20% missing is a problem and over 30% or so missing is a major problem. Now that Altair 5 has been released, `splink` charts are alt.Chart() objects with better documentation. Below, the missingness chart is displayed for the noisy dataset only because the original dataset has no missingness:

```
import altair as alt
c_noise = linker.missingness_chart("df_noise")
c_noise.save("c_noise.png")                                              ①
                                                                         ②
# c_noise = linker.missingness_chart("df_noise").spec
# png_noise = vlc.vegalite_to_png(vl_spec=c_noise, scale=2)
# with open("missingness_chart_noise.png", "wb") as f:
#     f.write(png_noise)
```

① Enable missingness chart of dataset `df_noise` (displayed below)
② Equivalent code in outdated `splink` 3.9.1 (no longer used)

The Profile charts are additionally problematic for PDF printing if there are many columns (variables) because it is a combined graph, with one graph per variable combined into one graph. PDF is better than HTML for printing, especially for wide output. The workaround used here is to select two variables for Figure 2 and two variables for Figure 3. The profile charts show the frequency distributions of the linkage variables. For example, common first names in principle should have less match weight. Frequency/Profile charts are useful for basic exploring and error checking. The profile charts seem more useful for preliminary analyses than for a final report. Here is an example of a profile chart for `unique_id` and `first_name`:

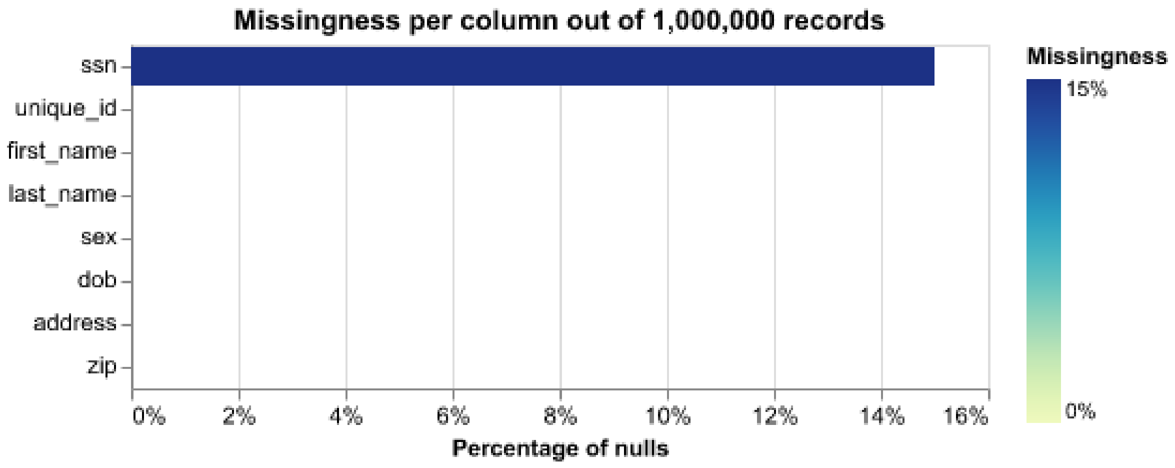

Figure 1: Missingness Chart of Noisy Dataset

```python
c_profile = linker.profile_columns(list(["unique_id", "first_name"]))  ①
c_profile.save("c_profile.png")
# The following resulted in 4 combined graphs, and too small text in splink 3.9.1:
# x_spec = linker.profile_columns(list(df.columns)).spec
```

① Profile chart of `unique_id` and `first_name`

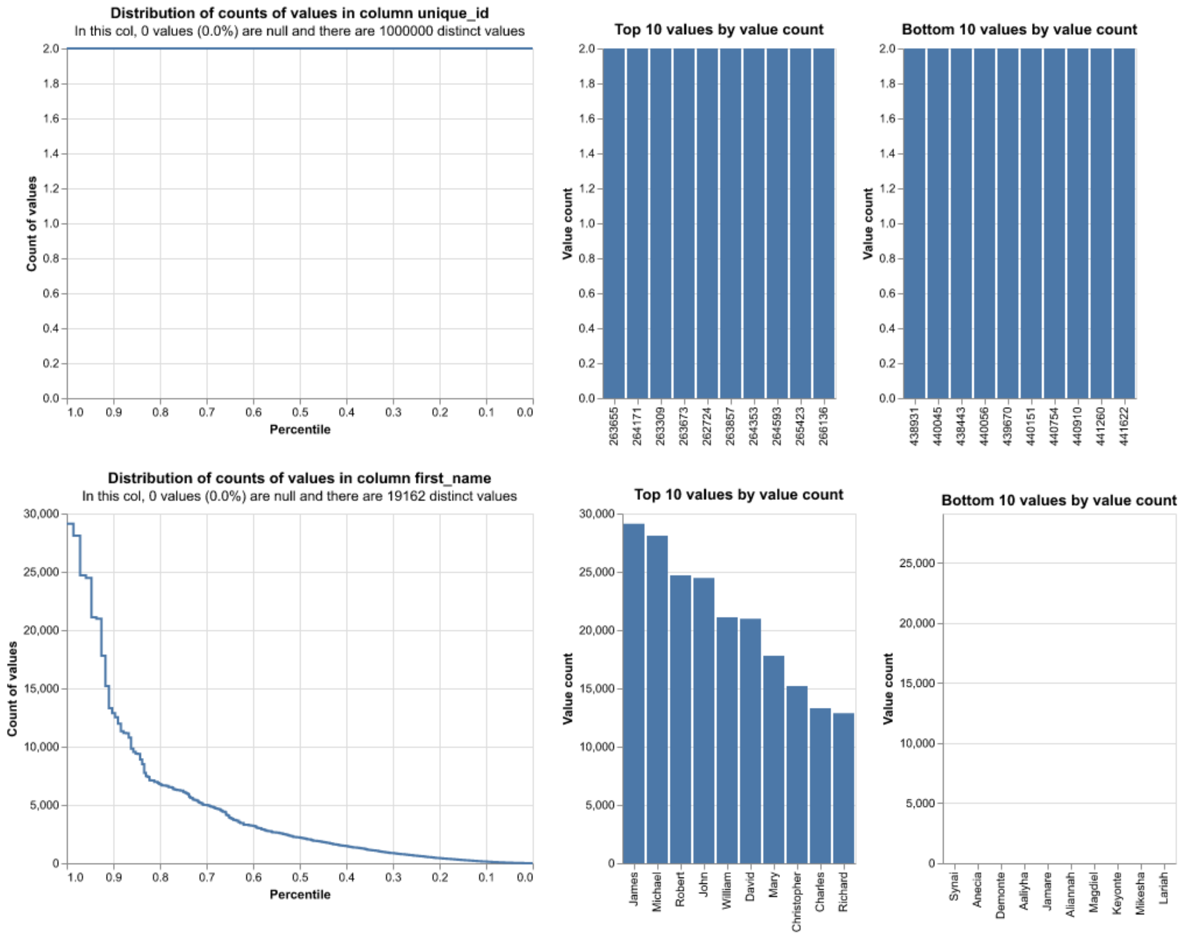

Figure 2: Profile Chart of unique_id and first_name

5.1 Estimate model parameters

There are three model parameters to be estimated in the linkage model: λ (`lambda`), `u` and `m`. All three parameters can be estimated using an EM algorithm. Typically, the first parameter to be estimated is the probability that two random records (with no blocking) match, `lambda`. It is also known as proportion of matches, link prevalence, and the starting value or prior for the EM algorithm. For example, if there are a million input records, as in the test data, and each has on average one match, then this value should be 1/1,000,000. In `splink`, the setting is probability_two_random_records_match. The default is 0.0001 (which sometimes is stated as 1e-04 or 1/10,000). The result is 999,753 links. The 100x lower value 1e-06 results in 287 fewer (that is, 999,466) links and the 100x higher value 0.01 (1e-02) results in 59 more (that is, 999,812) links. `lambda` is often greatly overestimated on small datasets, which is discussed in issue 462 and in discussion 1008.

To estimate `lambda`, instead of setting the value or of using the EM algorithm, we could use the linker function estimate_probability_two_random_records_match(). It is a direct estimation based on deterministic matching rules and a guess at "recall" (the true positive rate). A test run with the settings as in the `splink` (tutorial but with `ssn` instead of `email`) resulted in lambda 0.00188 (that is, 1.88-e5 or 188/100,000) and 44 fewer links (that is, in 999,709 links as compared with 999,753 links). We used the default setting 0.0001 since we have large datasets.

Typically, the second parameter to be estimated is the probability that false matches agree, `u`. It can be estimated directly using random sampling with the linker function `estimate_u_using_random_sampling()`. The linker functions for `lambda` and `u` could be run in either order since they are independent.

The third parameter to be estimated is the probability that true matches match, `m`. It is the most difficult parameter to estimate. Therefore, `m` (unlike `lambda` and `u`) can only be estimated using the EM algorithm. The `splink` linker function for the EM algorithm is `estimate_parameters_using_expectation_maximisation()`.

Each estimation pass requires the user to configure an estimation blocking rule to reduce the number of record comparisons generated to a manageable level. The first pass blocks on `first_name` and `last_name`. The second pass blocks on `zip` and first two letters of `last_name`. The final third pass blocks on `dob` and first two letters of `first_name`. The output is suppressed to save space, and because of code overflow (going off the page) in PDF output. See Quarto discussion 3693.

This match weight chart tells the user how each linkage variable was used. For example, the top graph shows that for `first_name` the most useful comparison was an exact match, as opposed to Jaro-Winkler and other comparisons. All variables have exact match as the most useful comparison for the test data. However, variable `sex` was of relatively little use – as shown by its low match weight (around 1) and light color (orange).

```
linker.estimate_u_using_random_sampling(target_rows=1e7, seed=123)     ①
print("--- %s seconds ---" % (time.time() - start_time))

# (The text is truncated in PDF. Could be resolved by editing the intermediate md file.)
training_blocking_rule = "l.first_name = r.first_name and l.last_name = r.last_name"  ②
training_session_names = linker.estimate_parameters_using_expectation_maximisation(   ③
    training_blocking_rule)

# Match Weights History charts
# (are not critical)
# training_session_names.match_weights_interactive_history_chart()

training_blocking_rule = """l.zip = r.zip and substr(l.last_name, 1,2) =
    substr(r.last_name, 1,2)"""                                                ④
training_session_names = linker.estimate_parameters_using_expectation_maximisation(
    training_blocking_rule)
training_session_names.match_weights_interactive_history_chart()

training_blocking_rule = """l.dob = r.dob and substr(l.first_name, 1,2) =
    substr(r.first_name, 1,2)"""                                               ⑤
training_session_names = linker.estimate_parameters_using_expectation_maximisation(
    training_blocking_rule)
training_session_names.match_weights_interactive_history_chart()
```

① Estimate the probability `u` that wrong matches match
② Blocking for estimation (training), first pass
③ Estimate the probability `m` that true matches match
④ Blocking for estimation (training), second pass
⑤ Blocking for estimation (training), third pass

```
c_matchweights = linker.match_weights_chart()                                  ①
c_matchweights.save("c_matchweights.png")
```

① Match Weight Chart

5.2 Predict results

The `splink` threshold match probability default is "None" which would return all record pairs, also non-links (non-matches). For the FCDS, 0.95 is a better default since we must return only matches (not non-matches).

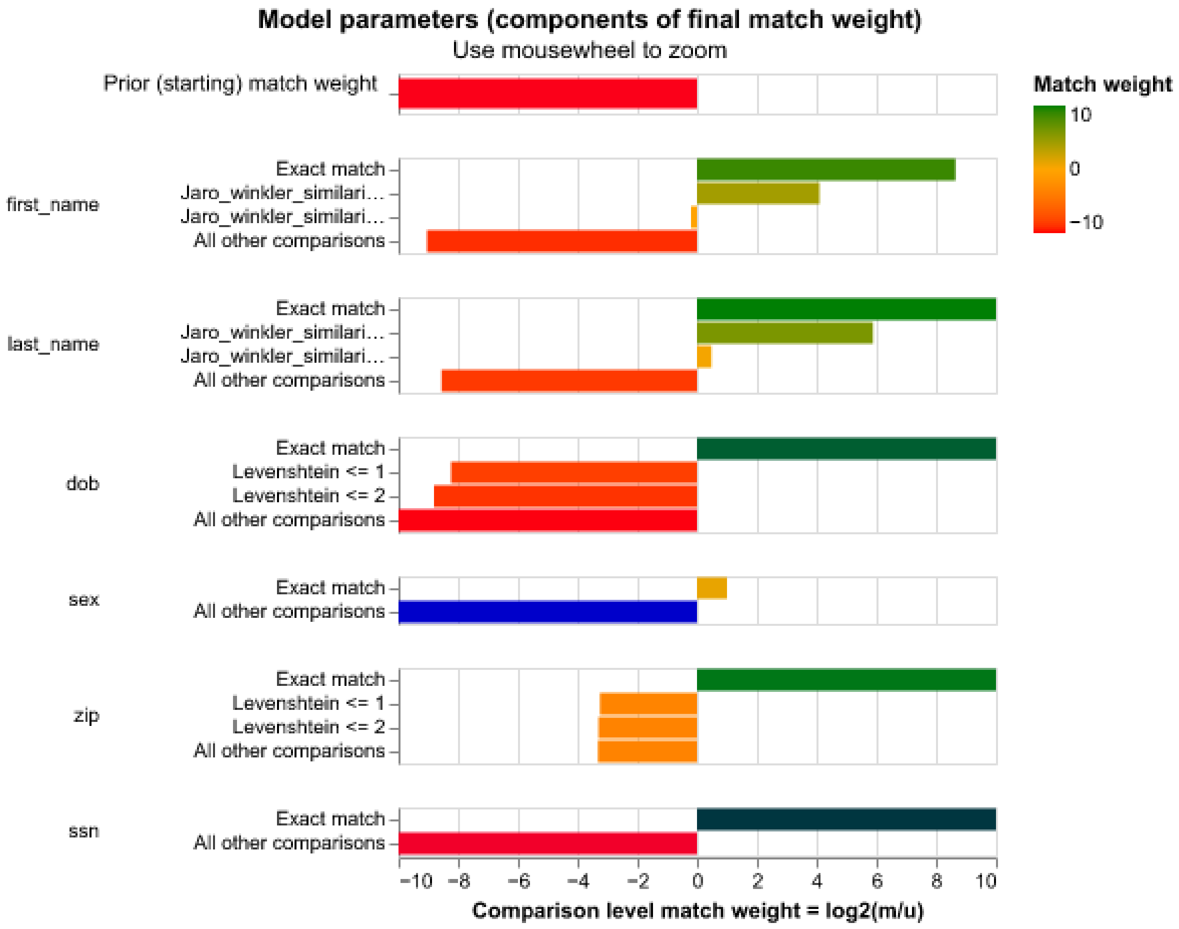

Figure 3: Match Weight Chart

```python
# Linker is a duckdb linker with link_type set to "link_only"
results = linker.predict(threshold_match_probability=0.95)          ①
records_to_plot = results.as_record_dict(limit=10)
```

① [Predict](), using the blocking rules in “step 2” and threshold 0.95, which records match.

To visualize predictions, we create a waterfall chart of matches. It is a graph of the final match weights. A negative value (a value below the horizontal line) means a likely “no match”. A positive value (a value above the horizontal line) means a likely “match”. The HTML version is interactive, so it can be used for clerical review. A separate `splink` tool for clerical review is being tested.

```python
c_waterfall = linker.waterfall_chart(records_to_plot, filter_nulls=False)   ①
                                                                            ②
# c_waterfall.layer[0].layer[1].encoding.x.axis.labelExpr = ''
c_waterfall.save("c_waterfall.png", scale_factor=2)                         ③
```

① Variable for waterfall chart
② No longer required with `vl-convert-python` version 0.11.1. See discussion 926
③ Save the waterfall chart using the new, simpler Altair 5 syntax

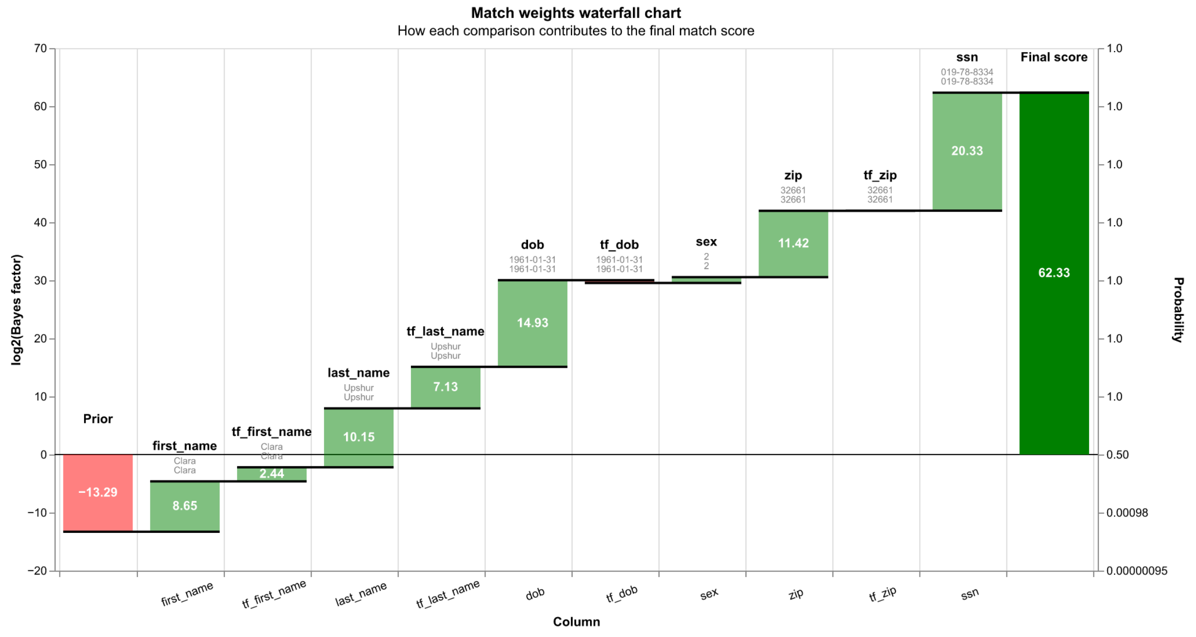

Figure 4: Waterfall Chart

Unlike `fastLink`, `splink` requires extra code for removing possible duplicates from the record linkage. The required extra code below is commented on little to save space, and to avoid confusion because de-duplication is not related to the linkage itself:

```
                                                                         ①
sql = f"""
with ranked as

(
select *,
row_number() OVER (
    PARTITION BY unique_id_l order by match_weight desc
    ) as row_number
from {results.physical_name}
)

select *
from ranked
where row_number = 1
"""
results = linker.query_sql(sql)                                          ②
```

① SQL "f-string" code
② Remove duplicates using SQL

There are three types of output variables:

- Results, 8 variables: `match_weight`, `match_probability`, `source_dataset_l`, `unique_id_l`, `source_dataset_r`, `unique_id_r`, `match_key`, `row_number`. The variable match_key refers to the blocking rule from which the comparisons was generated. The variable `row_number` always has value 1.
- Matching, 18 variables: All original variables with suffices `_l` and `_r` and the prefix gamma_ for comparisons. Examples are `first_name_l`, `first_name_r`, and `gamma_first_name`.
- Intermediate calculations, 18 variables: All variables with prefices tf_ for term frequency adjustment and bf_ for Bayes factor. Examples are `tf_first_name_l` and `bf_first_name`. Term frequency adjustment means adjusting for the skew/lack of uniformity in term frequencies, as discussed in a topic guide.

The table output from `results.head()` is right-truncated in PDF and therefore it is not used. Quarto is working on better Python table support, see issue 1716. As a workaround, we first use the Pandas DataFrame `shape` property to find the number of rows (observations) and columns (variables). The results dataset has 999,753 matches (that is, 247 missed matches) on 44 variables:

```python
results.shape
```

```
(999753, 44)
```

In the `fastLink` templates, Table 3 is frequencies of linkage pattern. A similar Table 3 in `splink` is possible but more difficult to interpret because in `splink` the `gamma_` variables can take on more values which make the table harder to interpret.

In the `fastLink` template, Table 4 is a confusion (truth) table. The `splink` developers are working on an interactive dashboard for labels, which in practice is required in `splink` for a confusion table. There is a tutorial for quality assurance of prediction results.

6 Step 4: Canonicalization

Step 4, canonicalization, is the post-processing. To view the dataframe for the clerical review until the new tool for clerical review is available, the easiest is to view it in R using `reticulate` ():

```r
library(reticulate)
View(py$results)
```

Usually, after the PRL, the FCDS subsets the dataset to the variables of interest. Here, we subset to four variables, namely, `unique_id_l`, `unique_id_r`, `match_weight`, and `match_probability`:

```python
cols_to_keep = ['unique_id_l', 'unique_id_r', 'match_weight', 'match_probability']  ①
results2 = results.loc[:, cols_to_keep]
results2.shape
```

① Reduce dataset to 4 (four) variables

```
(999753, 4)
```

Thereafter, for linkage data requests, the requested and approved FCDS cancer data are added. For these test data, this does not apply. Finally, the resulting data are saved into a CSV file. Here, we name the CSV dataset `results_monograph2023.csv` and we save only the four variables – not the index. The CSV dataset is about 42 MB. A sample of the results CSV would be smaller and easier to share. In many ways, Parquet files are better than other files such as CSV or Excel files. The largest issue with Parquet files, arguably, is that Stata and SAS cannot read and write them. The index is not needed for the results file. Only rarely do you need to save the index for better performance:

```
results2.to_csv('results_monograph2023.csv', index=False)    (1)
```

(1) Create CSV file

7 Results and Discussion

7.1 Comparative Summary Results on the Test Data

We used the most recent version of each PRL software, `Match*Pro` version 2.4 from June 14, 2023, `fastLink` version 0.6 from April 29, 2020, and `splink` version 3.9.2 from June 21, 2023. `Match*Pro` used the default frequency-based "EpiLink" weights (Contiero et al. 2005) versus EM. `fastLink` used 2 blocks (`sex`) versus 6 blocks (`sex` and `dob`). `splink` used lambda = 0.0001 (default) versus lambda = 0.01. Lambda is the probability that two random records match; see section 5.1 above for details.

Table 3. Performance of `Match*Pro`, `fastLink` and `splink`

Performance Category	`Match*Pro` 2.4 (Frequency/EM)	`fastLink` 0.6 (2 blocks/6 blocks)	`splink` 3.9.2 (Lambda = 0.0001 / 0.01)
Accuracy (possibles)	999,525 / 999,067	999,564 / 999,564	999,753 /**999,812**
Speed (hh:mm)	03:53 / 08:37	03:43 / 01:40	**00:02 / 00:02**
Transparency (see note)	Nonfree, current	**Free**, outdated	Open Source, **current**
User-friendliness	**GUI with User Guide**	Simple syntax, FCDS template	Difficult syntax, No template
Extensibility	No API, No reporting	R **API**, RStudio reporting	Python **API**, **Quarto reporting**
Recommended User/Use (Example)	**Any**/Simple (VPR Phase 1)	R/Intermediate (Data Requests)	Python/**Any** (Processing)

Note: Best results are marked in **bold**. Freedom refers to user-freedom, not to price.

Table 3 above summarizes the results of each PRL software on the artificial test data in terms of the five "ASTUE" components: Accuracy, Speed, Transparency, User-friendliness, and Extensibility. Table 3 reports two results for Accuracy and Speed (default versus modified), and one result otherwise. Usually, blocking leads both to faster results and some missed matches (FN). The accuracy did not worsen for `fastLink` on the test data because the blocking variable `dob` was without errors.

Table 4 below summarizes the FCDS view of the strengths and weaknesses of each PRL software as a scorecard with range 0-10 from worst to best. Both the "free" `fastLink` and the "open source" `splink` are scored 1 of 1 in terms of freedom, a category of Transparency. For the fall, the expected performance scores are `Match*Pro`=3, `fastLink`=8 and `splink`=8. The next performance comparison likely will focus on the accuracy and speed of `fastLink` and `splink` for linkage data requests.

Table 4. Performance Scorecard of `Match*Pro`, `fastLink` and `splink`

Performance Category	`Match*Pro` 2.4 (Frequency/EM)	`fastLink` 0.6 (Current/Expected)	`splink` 3.9.2 (Current/Expected)
Accuracy (0-2)	0/1	1/2	1/1
Speed (0-2)	1/0	1/2	2/2
Transparency (0-2)	0/0	1/2	2/2
User-friendliness (0-2)	2/2	1/1	0/1
Extensibility (0-2)	0/0	1/1	2/2
Total (0-10)	3/3	5/8	7/8

Note: Total score range is 0-10 with 0=Worst and 10=Best. The development plan of `Match*Pro` is unknown. The next release of `fastLink` will focus on accuracy (active learning) and on speed. The next releases of `splink` likely will focus on user-friendliness by documenting the new Altair 5 graphics and the new tool for clerical review.

Arguably, there are three possibly feasible and viable uses for `splink` at the FCDS. The uses, in order of suggested priority, are: 1) replace `fastLink` for linkage data requests, 2) reduce the amount of manual review for de-duplication, and 3) replace real-time deterministic record linkages with real-time PRL.

7.2 Discussion: Three possible uses of `splink` at the FCDS

The first possible use is to **replace `fastLink` for linkage data requests**. A linkage data request is a data request that involves linking internal FCDS data to an external data set. The FCDS will fill linkage data requests within 8 weeks once the request and cost have been approved. Currently, the FCDS prefers to use R and `fastLink` for linkage data requests. On average, the completion time for linkage data requests is about 3 weeks. Of those 3 weeks, approximately week 1 is for pre-processing (data cleaning) in R, week 2 is for the actual PRL using `fastLink`, and week 3 is for the manual review of of the possibly wrong matches (false positives). There is no "week 4" to extract cancer data. The cancer data are already part of the pre-processing when you extract the finder finder file from the FCDS database.

The potential advantage of replacing `fastLink` with `splink` for linkage data requests is much shorter completion time for the actual PRL and possibly shorter completion time for the manual review. This author estimates that about 33% or 1 week of 3 can be saved for each PRL in linkage data requests by replacing the current version of `fastLink` with `splink` once the FCDS has created a `splink` template.

The package `splink` adds a range of technical improvements, increased functionality and customisation options. `splink` completes a PRL of 1 million records in about one minute. In contrast, `fastLink` typically would need several hours for a similar linkage. In terms of accuracy, the results of `fastLink` and `splink` should be similar because they use the same underlying Fellegi-Sunter statistical model as documented for `fastLink`. However, `splink` adds several comparators and it allows a more flexible use of those comparators which often makes `splink` more accurate than `fastLink`. That is, complex models are often more accurate. Also, `splink` adds more tools for quality analysis such as waterfall charts.

There are few technology disadvantages with `splink` compared with `fastLink` for linkage data requests at the FCDS. The largest technical issue is that `splink` is rather unpolished in some areas. For example, removing duplicates in `splink` requires the use of SQL code whereas `fastLink` by default removes duplicates. Another example is that `splink`, unlike `fastLink`, requires labeled data for creating the confusion matrix. Quarto for Python is similarly unpolished compared with Quarto for R. Two examples are inline code and PDF output.

The economic feasibility of using `splink` instead of `fastLink` for linkage data requests is intertwined with the organizational feasibility. Both software are free and highly popular. The FCDS will need an FCDS template using Python/`splink`/Quarto Markdown, similar to the FCDS template using R/`fastLink`/R Markdown, to fully benefit from `splink`.

In general, `splink` primarily requires Python skills whereas `fastLink` primarily requires R skills. The differences between R markdown and Quarto markdown are relatively small. However, both Python/`splink` and R/`fastLink` are examples of technical hard skills. In terms of scheduling, if the Florida Department of Health Cancer Registry Program approves, the easiest would be to make "An FCDS template using Python/`splink`" the 2024 monograph deliverable. At the FCDS, currently only the author is capable of using Python/`splink` and R/`fastLink`. If the FCDS creates a new template for using Python/`splink`, in addition to the existing template for using and R/`fastLink`, then it will be easier for the FCDS management to decide whether to move forward with Python/`splink` or with R/`fastLink`.

A possible typology to summarize the current situation is the Rumsfeld matrix, which is a 2*2 matrix of knowns and unknowns.

1. "Known knowns" (things we are aware of and understand). Currently at the FCDS, depending on the person doing the PRL, it means a choice of `Automatch`, `fastLink` or `splink`, or `Match*Pro`.

2. "Known unknowns" (things we are aware of but do not understand). Examples are creating a `splink` template and someone else learning it and/or the `fastLink` template.

3. "Unknown knowns" (things we understand but are not aware of). Examples are promised new features of `splink` such as easier clerical review and better graphics documentation. The graphical user interface (GUI) for clerical review will be a stand-alone GUI. There is no plan to develop a complete GUI for `splink`. The company Nelson Scientific Labs is developing a GUI named `ShinyLink`. However, `ShinyLink` is not usable yet because it does not have blocking.

The `fastLink` developers are working on a new release for the fall which will be more accurate and faster. The improvement in accuracy will mostly come from "Active Learning" (Enamorado 2019) which is a semi-automated way of classifying record pairs that otherwise would require clerical review ("possibles"). The author expects about 1-2% improved accuracy and 5-10 times less clerical review thanks to active learning in `fastLink`. The next `fastLink` version is expected to have the Damerau-Levenshtein string distance comparator from the package `stringdist` for better partial matching of Social Security Number. `splink` uses Damerau-Levenshtein from `DuckDB`. The next version of `fastLink` will not use `DuckDB` because `DuckDB` does not perform well with input datasets over 2 million records. Instead, `fastLink` is expected to be 50% faster than now by using sampling to estimate model parameters. That would make the expected new run time on the test data about 50 minutes, or 25 times slower than `splink`.

4. "Unknown unknowns" (things we are neither aware of nor understand). Examples would be new unexpected features in `splink`.

An alternative to create a `splink` template for the 2024 monograph is to simply wait a year or two until it is a more compelling topic. Prediction is hard. In a year or two, it is likely that also `fastLink` and `Match*Pro` will have some new major features such as "probabilistic blocking" (Enamorado and Steorts 2020) or high-performance "GPU computing" for `fastLink` or an improved EM algorithm for `Match*Pro`.

The second use is to **reduce the amount of manual review for `Match*Pro` de-duplication**. NAACCR requires the use of `Match*Pro` for de-duplication of the annually submitted data. However, `Match*Pro` is expected to have about 1-2% more wrong matches (false positives) than `fastLink` and `splink`. Technologically, de-duplication is not more challenging than record linkage. The technological issue rather is scale, which means that the much faster and more accurate `splink` could be useful. A `splink` template for linkage data requests will have the spillover benefit to make `splink` easier to use for de-duplication. However, a `splink` template for de-duplication is a better technological solution if reducing the amount of manual review for de-duplication is the focus. In terms of the mentioned

Rumsfeld matrix, the issue is known unknowns. That is, we are aware of the required de-duplication issue but we do not have an efficient solution. Otherwise, the feasibility issues are very similar in the first and second uses.

The third possible use is to **replace real-time deterministic record linkages with real-time PRL**. R and `fastLink` is currently too slow to replace the real-time (also known as "spine-based") deterministic record linkages with real-time PRL. However, in terms technological feasibility, Python and `splink` can do real-time PRL. The `splink` Github webpage has a an example of real-time PRL. In general, this third use is the most high-risk and high-reward because it affects the operational processing, as opposed to linkage data requests in the first use or the annual de-duplication in the second use. Organizationally, this third use is harder because the FCDS relies on the consulting expertise of Advanced Consulting Enterprises. However, if this third use becomes a priority, then `splink` is a feasible solution.

Another possibly feasible improvement for this use is the Julia package SpineBasedRecordLinkage. However, the Julia package is only for real-time deterministic record linkages and to review it is outside of the scope for this feasibility study of `splink`.

8 Recommendation

This study shows that it is feasible to use `splink` at the FCDS, especially for replacing `fastLink` in linkage data requests, because `splink` is about 50 times faster and more accurate in large and complex linkages. However, for now, `fastLink` remains better overall than `splink` for the FCDS by having a template, much simpler syntax and much better Markdown support, and linkage results are summarized with tables that are easy to understand.

For the next budget year, 2023-2024, the author (Anders) recommends to continue to improve PRL at the FCDS. For now, the proposed topic of the 2024 monograph is titled "More Accurate Probabilistic Record Linkage using `fastLink` with Active Learning".

In order of priority, these are the specific recommendations:

- Improve the accuracy and speed of `fastLink` by using the expected new release.
- Improve the user-friendliness of `fastLink` by updating the FCDS templates from R Markdown to Quarto Markdown.
- Improve the user-friendliness of `fastLink` and `splink` by reducing the need for data cleaning. Decide if the "standardized" `STD_` variables in the PATIENT table can be used in the MV_DATAREQUEST table for linkage data requests.
- Work with Abraham Flaxman to improve and make public the FCDS test data. Related test data from the package `pseudopeople` are discussed for `splink` in discussion 1361.
- Backup plan if the `fastLink` release is delayed again: Begin to implement FCDS templates for using `splink`.

References

Ansolabehere, Stephen, and Eitan Hersh. 2017. "ADGN: An Algorithm for Record Linkage Using Address, Date of Birth, Gender, and Name." *Statistics and Public Policy* 4 (1): 1–10. https://doi.org/10.1080/2330443X.2017.1389620.

Binette, Olivier, and Rebecca C. Steorts. 2022. "(Almost) All of Entity Resolution." *Science Advances* 8 (12): 1–14. https://doi.org/10.1126/sciadv.abi8021.

Contiero, Paolo, Andrea Tittarelli, G. Tagliabue, A. Maghini, Sabrina Fabiano, P. Crosignani, and R. Tessandori. 2005. "The Epilink Record Linkage Software: Presentation and Results of Linkage Test on Cancer Registry Files." *Methods of Information in Medicine* 44 (1): 66–71. https://doi.org/10.1055/s-0038-1633924.

Enamorado, Ted. 2019. "Active Learning for Probabilistic Record Linkage." *SSRN e-Print*, 1–37. http://dx.doi.org/10.2139/ssrn.3257638.

———. 2021. "A Primer on Probabilistic Record Linkage." In *Handbook of Computational Social Science, Volume 2*, edited by Uwe Engel, Anabel Quan-Haase, Sunny Xun Liu, and Lars Lyberg, 95–107. Routledge. https://doi.org/10.4324/9781003025245-8.

Enamorado, Ted, Bejamin Fifield, and Kosuke Imai. 2019. "Using a Probabilistic Model to Assist Merging of Large-scale Administrative Records." *American Political Science Review* 113 (2): 353–71. https://doi.org/10.1017/S0003055418000783.

Enamorado, Ted, and Rebecca C. Steorts. 2020. "Probabilistic Blocking and Distributed Bayesian Entity Resolution." In *Privacy in Statistical Databases*, edited by Domingo-Ferrer J. and Muralidhar K., 224–39. Cham, Switzerland: Springer Lecture Notes in Computer Science. Volume 12276. https://doi.org/10.1007/978-3-030-57521-2_16.

Linacre, Robin, Sam Lindsay, Theodore Manassis, Zoe Slade, and Tom Hepworth. 2022. "Splink: Free Software for Probabilistic Record Linkage at Scale." *International Journal of Population Data Science* 7 (3): 23. https://doi.org/10.23889/ijpds.v7i3.1794.